// Computer Program Listing Appendix Under 37 CFR 1.52(e)
// Code segments.txt

The triggers are checked at regular intervals. When it is time for
them to be checked, the SWM daemon calls this function. It first
flushes all cached built-in variables from lua and then iterates
through the active jobs, checking the triggers for each in turn.
\begin{code} *&*/
```
static void swm_triggers_check_on_timer(void *dummy) {
  if (operation_mode != SWM_HALT_MODE) {
    sychron_eval_builtin_flush_cache(lua_state);
    swm_rules_app_iterator(lswm_triggers_check_for_one_job);
  }
  trigger_check_step++;
  return;
}
```
/*&* \end{code}
This is the code that will called by the job iterator for a single
job. It evaluates the triggers for each job and decides if any action
needs to be taken.
Note that when the lua evaluator is called we check the period for any
builtin variable calculations, supply no changed variables and
reconstruct the function name for the condition.
\begin{code} *&*/
```
static void lswm_triggers_check_for_one_job(swm_job_id_t job_id,
                                            int rule_ind) {
  swm_sjd_trigger_t *trigger = NULL;
  syc_uint32_t i, step_freq;
  swm_cause_t cause = {"ON-TIMER policy", 0.0};
  /*-- 1. iterate through all triggers for this rule --*/
  for (i = 0; i < rule_set->rules[rule_ind].triggers.ntriggers; i++) {
    trigger = &rule_set->rules[rule_ind].triggers.trigger[i];
    /*-- 1a. only consider the desired triggers --*/
    if (trigger->condition.check == SWM_SJD_TRIGGER_CHECK_ON_TIMER) {
      step_freq =
       trigger->condition.timer ?
       ((trigger->condition.timer + swm_triggers_time_interval - 1) /
         swm_triggers_time_interval) :
       SWM_TRIGGER_CONDITION_DEFAULT_FREQUENCY;
      if ((trigger_check_step % step_freq) == 0)
        lswm_triggers_condition_action(job_id,rule_ind,i,trigger,
                                       trigger->condition.timer,"",&cause);
    }
  }
  return;
}
```
/*&* \end{code}
The next block of code ensures the one-off evaluation of policies of "ON-SET" policies (i.e., polices evaluated when a user-defined variable used in their condition changes its value):
When a job variable is set we need to evaluate all trigger conditions

```
for the job that are of type \emph{evaluate on set}.
\begin{code} *&*/
static int lswm_triggers_variable_set(swm_job_id_t job_id,
                                      char *variable,
                                      double value,
                                      int rule_ind) {
  swm_sjd_triggers_t *triggers = &rule_set->rules[rule_ind].triggers;
  swm_sjd_trigger_t *trigger;
  syc_uint32_t i;
  int err, any_err = 0;
  swm_cause_t cause;
  /*-- 0. Initialise cause --*/
  cause.name = variable;
  cause.value = value;
  /*-- 1. iterate through triggers for this job --*/
  for (i = 0; i < triggers->ntriggers; i++) {
    trigger = &triggers->trigger[i];
    if (trigger->condition.check == SWM_SJD_TRIGGER_CHECK_ON_SET) {
      err = lswm_triggers_condition_action(job_id,rule_ind,i,
                                           trigger,0,variable,&cause);
      if (err)
        any_err = err;
    }
  }
  return any_err;
}
/*&* \end{code}
Check a conditional and perform an action if necessary. The function
returns zero on sucess, and a negative value on error.
\begin{code} *&*/
static int lswm_triggers_condition_action(swm_job_id_t       job_id,
                                          int                rule_ind,
                                          syc_uint32_t       cond_ind,
                                          swm_sjd_trigger_t *trigger,
                                          syc_uint32_t       timer,
                                          const char        *variable,
                                          const swm_cause_t *cause) {
  int          res  = 0;
  char        *name = NULL;
  syc_uint32_t period, flags;
  const char  *lua_result = NULL;
  sychron_eval_transition_t trans_result = no_evaluation;
  double eval_result;
  time_t atomicity_keepout = trigger->action.atomicity;
  /* Do not evaluate the trigger if the action has happened recently */
  if (trigger->action.atomicity &&
      !lswm_triggers_atomicity_maybe_ok(job_id,rule_ind,cond_ind,
                                        atomicity_keepout))
    return 0;
  period = trigger->eval_period ?
```

```
        trigger->eval_period : SWM_VARIABLES_BUILT_IN_DEFAULT_PERIOD;
  name = lswm_make_lua_name(rule_ind, cond_ind, 1);
  if (name) {
    flags = SYCHRON_EVAL_FLAGS_LOGGING;
    if (trigger->action.when == SWM_SJD_TRIGGER_ON_TRANSITION) {
      flags |= SYCHRON_EVAL_FLAGS_TRANSITION;
    }
    lua_result = sychron_eval_function(lua_state, name, job_id,
                                       period, timer, variable, flags,
                                       &eval_result, &trans_result);
    /*-- if result matches trigger specification take action --*/
    if (lua_result) {
      slog_msg(SLOG_DEBUG, "Error evaluating trigger expression "
               "[lswm_triggers_variable_set(): %s] "
               "JOB ID %d EXPRESSION INDEX %d",
               lua_result, (int)job_id, cond_ind);
      res = -1;
    }
    else {
      if ((trigger->action.when == SWM_SJD_TRIGGER_ON_TRUE &&
           eval_result != 0.0) ||
          (trigger->action.when == SWM_SJD_TRIGGER_ON_TRANSITION &&
          trans_result == transition_to_true)) {
        /* Only evaluate trigger if the action has not happened recently */
        if (!trigger->action.atomicity ||
            lswm_triggers_atomicity_commit(job_id,rule_ind,cond_ind,
                                           atomicity_keepout))
          lswm_triggers_perform_action(job_id, rule_ind, cond_ind,
                                       eval_result, cause);
      }
    }
  }
  else {
    slog_msg(SLOG_WARNING, "Error creating expression name "
             "[lswm_triggers_variable_set()] "
             "JOB ID %d EXPRESSION INDEX %d", (int)job_id, cond_ind);
    res = -1;
  }
  /* free memory allocated for name */
  free(name);
  return res;
}
/*&* \end{code}
The trigger conditions are evaluated whenever a new variable is set,
or the timer expires. When a trigger action needs to be performed the
following function is called.
\begin{code} *&*/
static void lswm_triggers_perform_action(swm_job_id_t job_id,
                                         int          rule_ind,
                                         syc_uint32_t trigger_ind,
```

```
                                double       eval_result,
                                const swm_cause_t *cause) {
int i;
char *name = NULL;
syc_uint32_t period;
swm_app_rule_t *job_rule;
swm_sjd_trigger_t *trigger;
swm_sjd_trigger_condition_t *condition;
swm_sjd_trigger_action_t *action;
job_rule    = &rule_set->rules[rule_ind];
trigger     = &job_rule->triggers.trigger[trigger_ind];
action      = &trigger->action;
condition   = &trigger->condition;
/*-- 1. SLA action --*/
if (action->type == SWM_SJD_TRIGGER_SLA) {
  /*-- Is the default/reference SLA being reinstated? --*/
  if (action->action.sla.type == SWM_SJD_TRIGGER_SLA_DEFAULT) {
    for (i = 0; i < job_rule->sla_rules.nrules; i++)
      if (job_rule->sla_rules.rule[i].resource ==
            action->action.sla.sla_rule.resource &&
          job_rule->sla_rules.rule[i].index ==
            action->action.sla.sla_rule.index) {
        swm_jobs_adjust_sla_rules(job_id, action->action.sla.type,
                                  &job_rule->sla_rules.rule[i],
                                  condition->check ==
                                    SWM_SJD_TRIGGER_CHECK_ON_SET,
                                  action->action.sla.nservers,
                                  cause);
        break;
      }
  }
  /*-- Otherwise, the current SLA is being adjusted --*/
  else {
    name = lswm_make_lua_name(rule_ind, trigger_ind, 0);
    if (name) {
      double sla_pool_value;
      sychron_eval_transition_t trans_result = no_evaluation;
      const char *result = NULL;
      period = trigger->eval_period ?
        trigger->eval_period : SWM_VARIABLES_BUILT_IN_DEFAULT_PERIOD;
      /*-- 1a. evaluate SLA expression --*/
      result = sychron_eval_function(lua_state, name, job_id,
                                     period, 0, "",
                                     SYCHRON_EVAL_FLAGS_NONE,
                                     &sla_pool_value, &trans_result);
      if (result) {
        /* failed evaluation */
        slog_msg(SLOG_WARNING, "Error evaluating sla expression "
                 "[lswm_triggers_perform_action(): %s] "
                 "JOB ID %d EXPRESSION INDEX %d -- no action taken",
```

```
                result, (int)job_id, trigger_ind);
        }
        else {
          char *tmp_expr;
          char pool_value[32];
          /*-- 1b. successful evaluation, set new sla value */
          snprintf(pool_value, sizeof(pool_value),
                "%d", (int) sla_pool_value);
          tmp_expr = action->action.sla.sla_rule.values.pool_value.amount;
          action->action.sla.sla_rule.values.pool_value.amount =
            pool_value;
          swm_jobs_adjust_sla_rules(job_id, action->action.sla.type,
                                &action->action.sla.sla_rule,
                                condition->check ==
                                  SWM_SJD_TRIGGER_CHECK_ON_SET,
                                action->action.sla.nservers,
                                cause);
          action->action.sla.sla_rule.values.pool_value.amount = tmp_expr;
        }
      }
      else {
        slog_msg(SLOG_WARNING, "Error creating sla expression name "
              "[lswm_triggers_perform_action()] "
              "JOB ID %d EXPRESSION INDEX %d -- no action taken",
              (int)job_id, trigger_ind);
      }
      /* free memory allocated for name */
      free(name);
    }
  }
  /*-- 2. SCRIPT ACTION --*/
  else if (action->type == SWM_SJD_TRIGGER_SCRIPT) {
    name = lswm_make_lua_name(rule_ind, trigger_ind, 0);
    lswm_triggers_script_action(job_id,
                            name,
                            job_rule,
                            trigger,
                            eval_result);
    free(name);
  }
  /*-- 3. LB ACTION --*/
  else if (action->type == SWM_SJD_TRIGGER_LB) {
    /* adjust lb-params and the weight of the lb-params current device */
    swm_jobs_adjust_lb_rules(job_id,
                          action->action.lb.type,
                          crt_lb_method.name,
                          &action->action.lb.lb_params);
  }
  return;
}
```

/*&* \end{code}